DISTRIBUTED COMPUTING

Unit – 1

Chapter 1 - Characterization of D.S.

1. **Define Distributed System & discuss its characteristics. Give examples for Distributed Systems.**

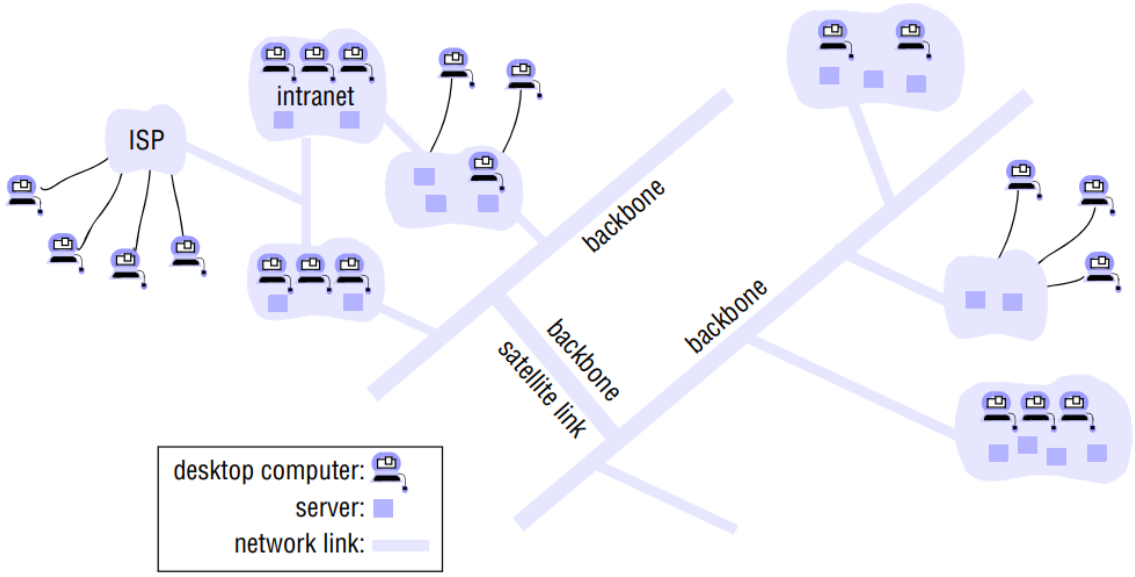
A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.

Characteristics:

- **Concurrency:** Concurrent programs execution is the norm. I can do my work on my computer while you do your work on yours, sharing resources such as web pages or files when necessary. Resources can be added or removes easily.
- **No global Clock:** Close coordination often depends on a shared idea of the time at which the programs' actions occur. But it turns out that there are limits to the accuracy with which the computers in a network can synchronize their clocks – there is no single global notion of the correct time.
- **Independent Failure:** Faults in the network result in the isolation of the computers that are connected to it, but that doesn't mean that they stop running.

Examples of D.S. (can explain any one):
Internet, Intranet, Mobile and ubiquitous computing.

Internet:

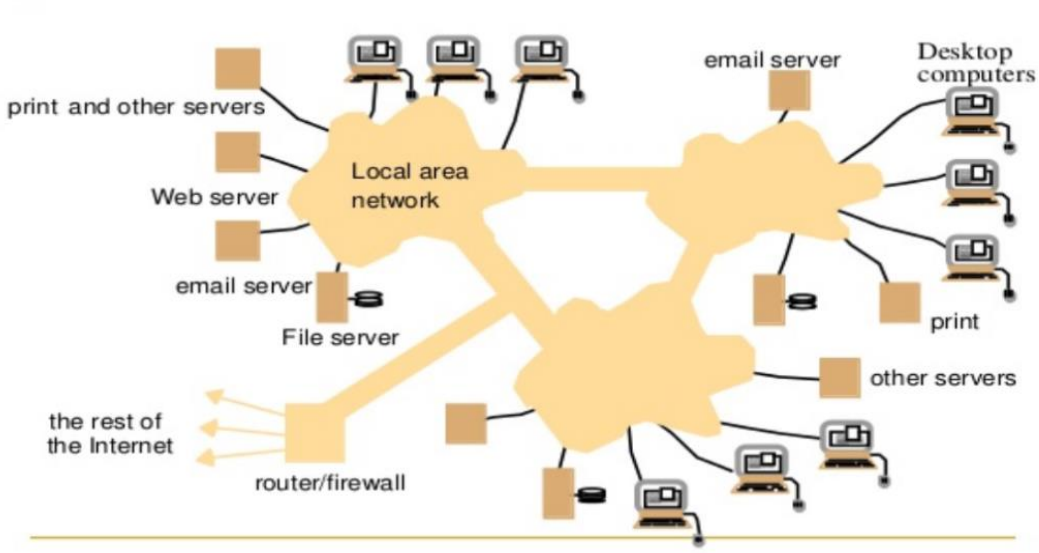


A typical portion of the Internet

The Internet is a very large distributed system. It enables users, wherever they are, to make use of services like www, email, file transfer, etc. The set of services is open-ended i.e. it can

be extended by the addition of server computers and new types of services. The modern Internet is a vast interconnected collection of computer networks of many different types, with a range of types increasing all the time.

Intranet:

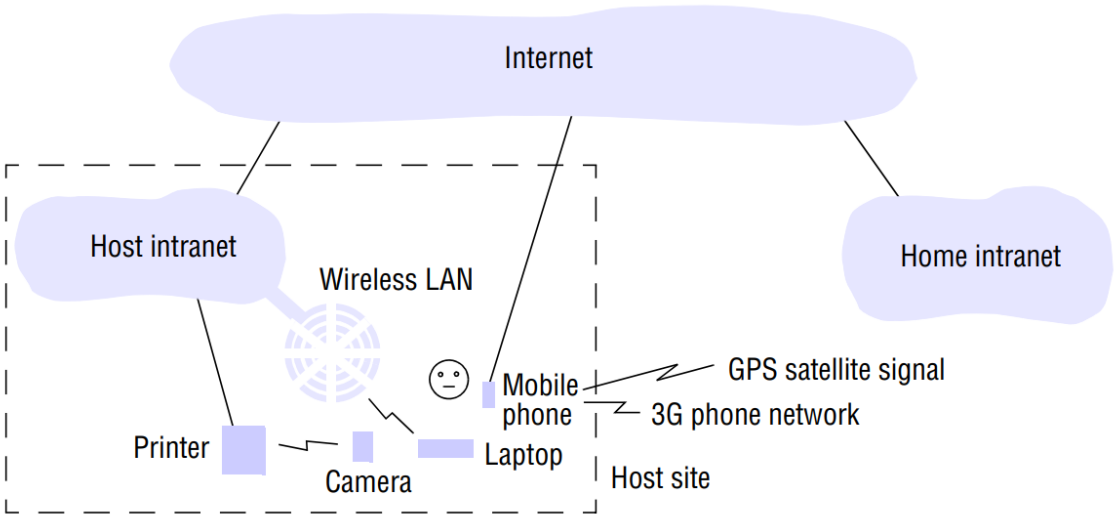


A typical Intranet

- An Intranet is a portion of the internet that is separately administered and has a boundary that can be configured to enforce local security policies.
- It may be composed of several LANs linked by backbone connections.
- The network configuration of a particular intranet is the responsibility of the organization that administers it.
- An intranet is connected to the Internet via a router, which allows the users to use the services available on the Internet.
- A firewall is used to protect the intranet by preventing unauthorized messages from leaving or entering.
- Some organizations do not wish to connect their internal networks to the Internet at all.
- E.g., police and other security and law enforcement agencies are likely to have at least some internal networks that are isolated from the outside world.
- These organizations can be connected to the Internet to avail themselves of the services by dispensing with the firewall.
- The main issues arising in the design of components for use in intranets are:
    - File services are needed to enable users to share data
    - Firewalls should ensure legitimate access to services
    - The cost of installation and support should be minimum

Mobile and Ubiquitous Computing:

- Integration of portable computing devices like Laptops, smartphones, handheld devices, pagers, digital cameras, smartwatches, devices embedded in appliances like refrigerators, washing machines, cars, etc. with the distributed systems became possible because of the technological advances in device miniaturization and wireless networking.
- These devices can be connected to each other conveniently in different places, making mobile computing possible.
- In mobile computing, users who are away from the home intranet, are still allowed to access resources via the devices they carry.
- Ubiquitous computing is the harnessing of many small, cheap computational devices that are present in users' physical environments, including home, office, and others.
- The term ubiquitous is intended to suggest that small computing devices will eventually become so pervasive in everyday objects that they are scarcely noticed. The presence of computers everywhere is useful only when they can communicate with one another.
- E.g., it would be convenient for users to control their washing machine or their entertainment system using a “Universal remote control” device at home.
- The mobile user can get benefit from computers that are everywhere.
- Ubiquitous computing could benefit users while they remain in a single environment such as the home, office, or hospital.
- The figure shows a user who is visiting a host organization. The user's home intranet and the host intranet at the site that the user is visiting.
- Both intranets are connected to the rest of the Internet.



2. List the challenges in distributed systems. Explain in detail any two of them.

- Heterogeneity:
  - The Internet enables users to access services and run applications over a heterogeneous collection of computers and networks.
- Openness:
  - The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways.
  - The openness of distributed system determined primarily by the degree to which new resource sharing devices can be added

- Security:
  - Their security is therefore of considerable importance.
  - Security for information resources has three components:
    - confidentiality (protection against disclosure to unauthorized individuals),
    - Integrity (protection against alteration or corruption), and
    - availability (protection against interference with the means to access the resources).
- Scalability:
  - A system is described as scalable if it will remain effective when there is a significant increase in the number of resources and the number of users.
- Failure Handling:
  - Failures in a distributed system are partial – that is, some components fail while others continue to function.
  - The following are techniques for dealing with failures. ▪ Detecting failures ▪ Masking failures
    - Tolerating failures ▪ Recovery from failure ▪ Redundancy
- Concurrency:
  - Both services and applications provide resources that can be shared by clients in a distributed system.
  - There is therefore a possibility that several clients will attempt to access a shared resource at the same time.
- Transparency:
  - Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components.

Chapter 2 – System Models

3. **Define Architecture Model. Mention its goal & explain the following with an example.**
   - **i) Mobile Code**
   - **ii) Mobile Agent**
   - **iii) Proxy Server & Cache**
   - **iv) Peer-Peer network**

Ans:

Architectural model is concerned with the placement if its components and the relationships between them. The architecture of a system is its structure in terms of separately specified components and their interrelationships. The overall goal is to ensure that the structure will meet present and likely future demands on it. Major concerns are to make the system reliable, manageable, adaptable and cost-effective.

i) Mobile Code:

• It is used to refer to code that can be sent from one computer to another and run at the destination. • Example: java applets

ii) Mobile agent:

• Mobile agent is a running program that travels from one computer to another carrying out a task to someone's behalf, such as collecting information, eventually returning with the results. (e.g.: Google form)

• Mobile agent is a complete program (including both code & data) that can work independently.

• Mobile agent can invoke local resources/data.

iii) Proxy & cache:

• proxy servers are used to increase availability and performance of the services by reducing the load on the network and web-server.

• Proxy server provides copies(replications) of resources which are managed by another server Cache: A store of recently used data objects that is closer to the client process than those remote objects

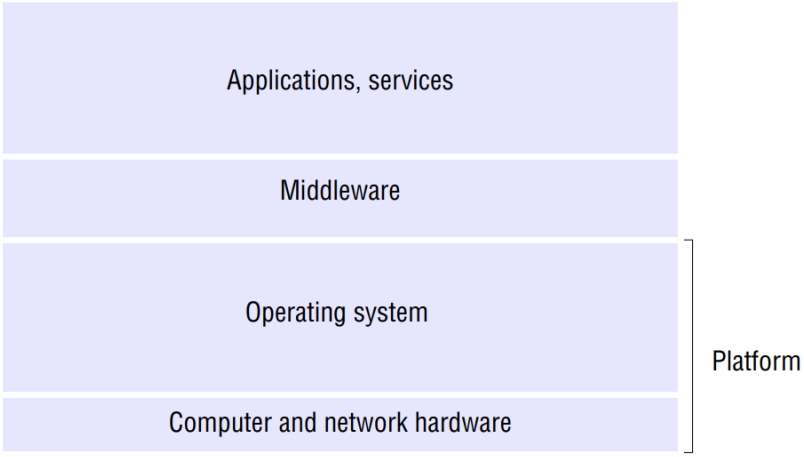
iv) Peer-Peer Network

- In this architecture all of the processes involved in a task or activity play similar roles, interacting cooperatively as peers without any distinction between client and server processes or the computers on which they run.
- In practical terms, all participating processes run the same program and offer the same set of interfaces to each other.
- While the client-server model offers a direct and relatively simple approach to the sharing of data and other resources, it scales poorly.

Example: BitTorrent file-sharing system.

4. Discuss the Software Layers of distributed system architectural model.

Ans:



- Application Layer: This is the topmost layer and the interface between the user and the system. It defines the application's functionalities and services that the user can access. Applications can include things like web browsers, email clients, and other software that users interact with directly.
- Service Layer: This layer provides the underlying services that applications use to perform their functions. Services can include things like databases, message queues, and other software that applications rely on to operate.
- Middleware Layer: This layer acts as a bridge between the application and service layers, allowing them to communicate and interact with one another. Middleware can include things like message brokers, remote procedure call (RPC) libraries, and other software that facilitates communication between different parts of the system.
- Operating System Layer: This layer provides the underlying infrastructure that the system runs on. The operating system provides services such as process management, file system management, and network communication.
- Computer Hardware Layer: This layer includes the physical hardware that the system runs on, such as servers, storage devices, and other computer hardware.
- Network Hardware Layer: This layer includes the physical hardware that connects the different components of the system, such as routers, switches, and other network devices. This layer is responsible for the physical transmission of data across the network, including issues such as addressing, routing and forwarding.

Each layer of the distributed system architectural model is responsible for a specific set of functions and interacts with the layers above and below it to provide the overall functionality of the system.

5. Describe the interaction model of distributed system.

Ans:

Interaction models are for handling time i.e. for process execution, message delivery, clock drifts, etc.

There are two variants of the Interaction Model

- Synchronous Distributed Systems:
  - The time taken to execute each step of a process has known lower and upper bounds.
  - Each message transmitted over a channel is received within a known bounded time.
  - Each process has a local clock whose drift rate from real-time has a known bound.
  - Ex., Client-server: request-response pattern, Shared-memory: locks, semaphores

- Asynchronous Distributed Systems:
  - The time taken to execute each step of a process can be arbitrarily long.
  - Each message transmitted over a channel may be received after an arbitrarily long time.
  - Each process has a local clock whose drift rate from real-time may be arbitrarily long.
  - Ex., Peer-to-peer: gossip protocol, Message passing: publish-subscribe pattern

6. Describe the failure model of distributed system.

Ans:

- Failures can occur both in processes and communication channels. The reason can be both software and hardware faults.
- Fault models are needed in order to build systems with predictable behaviour in case of faults (systems that are fault-tolerant).
- Such a system will function according to the predictions, only as long as the real faults behave as defined by the “fault model”.

Variants of the Failure Model

- ❖ Omission Failures:
  - ➢ A process or channel fails to perform the actions that it is supposed to do.
  - ➢ Process Omission Failure:
  - ➢ A process completes a send-operation, but the message is not put into the outgoing message buffer.
  - ➢ A message is put into a process's incoming message buffer, but the process does not receive it.
- ❖ Communication Omission Failure:
  - ➢ It is a 'fail-stop' failure, which means the server only exhibits crash failures, but at the same time, any correct server in the system can detect that this particular server has failed.

- Arbitrary Failures:
  - The term arbitrary or Byzantine failure is used to describe the worst possible failure semantics, in which any type of error may occur.
  - An arbitrary failure of a process is one in which it arbitrarily omits intended processing steps or takes unintended processing steps.
  - Process or channel may exhibit arbitrary behaviour when failing:
  - To send/receive arbitrary messages at arbitrary intervals.
  - A process may halt or perform "faulty" steps.

- Timing Failures:
  - Timing failures are applicable in synchronous distributed systems where time limits are set on process execution time, message delivery time, and clock drift rate.
  - Timing Failures can be classified as:

Class of failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

- Masking Failures:
  - Knowledge of the failure characteristics of a component can enable a new service to be designed to mask the failure of the components on which it depends.
  - A service masks a failure either by hiding it altogether or by converting it into a more acceptable type of failure.
  - Example: Multiple servers that hold replicas of data can continue to provide a service when one of them crashes.

7. Summarize the following design requirements for Distributed Architectures;
   i) Performance Issues
   ii) Quality of Service

Designing a distributed system requires careful consideration of various factors to ensure that it performs well and provides the necessary guarantees around service availability, reliability, and consistency.

- Performance Issues: One of the key design requirements for distributed systems is addressing performance concerns. This includes ensuring high throughput and low latency, as well as scalability, so that the system can handle increasing load and numbers of users. Techniques such as load balancing, caching, and partitioning data across multiple nodes can be used to improve performance.

- Quality of Service: Another important design requirement for distributed systems is providing guarantees around Quality of Service (QoS). This includes ensuring service availability, reliability, and consistency. To achieve this, fault tolerance mechanisms such as replication and redundancy can be implemented to ensure that the system can continue to operate in the event of failures. Consensus algorithms such as Paxos and Raft can also be used to ensure that all nodes have a consistent view of the data.

In addition to these, security, ease of management, and ease of integration with other systems are also important factors to consider while designing distributed systems. Overall, designing a distributed system requires careful consideration of various factors to ensure that it performs well, is reliable, and can scale to meet the needs of the users.

Unit – 2

Chapter 3 - IPC.

1. Explain the characteristics of IPC.

Ans:

- Synchronous and Asynchronous Communication:
  - In the synchronous form of communication, the sending and receiving processes synchronize with every message. In this case, both send and receive are blocking operations. Ex: Continuous chatting.
  - In the asynchronous form of communication, the use of the send operation is non-blocking and the receive operation can have blocking and non-blocking variants. Ex: WhatsApp.
- Message Destinations:
  - A local port is a message destination within a computer, specified as an integer.
  - A port has exactly one receiver but can have many senders.
  - Processes may use multiple ports to receive messages.
- Reliability:
  - A reliable communication is defined in terms of validity and integrity.
  - A point-to-point message service is described as reliable if messages are guaranteed despite a reasonable number of packets being dropped or lost.
  - For integrity, messages must arrive uncorrupted and without duplication.
- Ordering:
  - Some applications require that messages to be delivered in sender order.

2. Compare & Contrast between Synchronous & Asynchronous communication in the context of IPC.

Ans:

A queue is associated with each message destination. Sending processes cause messages to be added to remote queues and receiving processes remove messages from local queues. Communication between the sending and receiving processes may be either synchronous or asynchronous.

Synchronous:

In the synchronous form of communication, the sending and receiving processes synchronize at every message. In this case, both send and receive are blocking operations. Whenever a send is issued the sending process (or thread) is blocked until the corresponding receive is issued. Whenever a receive is issued by a process (or thread), it blocks until a message arrives.

Asynchronous:

In the asynchronous form of communication, the use of the send operation is nonblocking in that the sending process is allowed to proceed as soon as the message has been copied to a local buffer,

and the transmission of the message proceeds in parallel with the sending process. The receive operation can have blocking and non-blocking variants. In the non-blocking variant, the receiving process proceeds with its program after issuing a receive operation, which provides a buffer to be filled in the background, but it must separately receive notification that its buffer has been filled, by polling or interrupt.

3. Analyze the failure model of Request/Reply protocol in client-server Communication using UDP.

Ans:

- When *doOperation*, *getRequest* and *sendReply* are implemented over UDP datagrams, then they suffer from communication failures. That is:
  - They suffer from omission failures.
  - Messages are not guaranteed to be delivered in sender order.
- The protocol can suffer from the failure of processes. We assume that processes have crash failures. That is, when they halt, they remain halted – they do not produce Byzantine behaviour.
- To allow for occasions when a server has failed or a request or reply message is dropped, *doOperation* uses a timeout when it is waiting to get the server's reply message. The action taken when a timeout occurs depends upon the delivery guarantees being offered.

A. Timeouts

- There are various options as to what *doOperation* can do after a timeout. The simplest option is to return immediately from *doOperation* with an indication to the client that the *doOperation* has failed.
- To compensate for the possibility of lost messages, *doOperation* sends the request message repeatedly until either it gets a reply or it is reasonably sure that the delay is due to lack of response from the server rather than to lost messages. Eventually, when *doOperation* returns, it will indicate to the client by an exception that no result was received.

B. Discarding duplicate request messages

- In cases when the request message is retransmitted, the server may receive it more than once. For example, the server may receive the first request message but take longer than the client's timeout to execute the command and return the reply.
- This can lead to the server executing an operation more than once for the same request. To avoid this, the protocol is designed to recognize successive messages (from the same client) with the same request identifier and to filter out duplicates.

C. Lost reply messages

- If the server has already sent the reply when it receives a duplicate request it will need to execute the operation again to obtain the result, unless it has stored the result of the original execution.
- Some servers can execute their operations more than once and obtain the same results each time.
- An idempotent operation is an operation that can be performed repeatedly with the same effect as if it had been performed exactly once. A server whose operations are all idempotent need not take special measures to avoid executing its operations more than once.

4. Discuss issues relating to datagram communication.

Ans:

- **Message size:** The receiving process needs to specify an array of bytes of a particular size in which to receive a message. If the message is too big for the array it is truncated on arrival. The underlying IP protocol allows packet length of up to 216bytes which include the headers as well as the message. However, most environments impose a size restriction of 8 KB. Any application requiring messages larger than the maximum must fragment them into chunks of that size.
- **Blocking:** Sockets normally provide non-blocking sends and blocking receives for datagram communication. The send operation returns when it has handed the message to the underlying UDP and IP protocols which are responsible for transmitting it to its destination. The method receive blocks until a datagram is received unless a timeout has been set on the socket. If the process that involves the receive method has other work to do while waiting for the message, it should arrange to use a separate thread.
- **Timeouts:** The receive that blocks forever is suitable for use by a server that is waiting to receive requests from its clients. But in some programs, it is not appropriate that a process that has invoked a receive operation should wait indefinitely in situations where the sending process may have crashed or the expected message may have been lost. To allow for such requirements, timeouts can be set on sockets. Choosing an appropriate timeout interval is difficult, but it should be fairly large in comparison with the time required to transmit a message.
- **Receive from Any:** The receive method does not specify an origin for messages. Instead, an invocation of receive gets a message addressed to its socket from any origin. The receive method returns the internet address and local port of the sender, allowing the recipient to check where the message came from. It is possible to connect a datagram socket to a particular remote port and internet address, in which case the socket is only able to send and receive messages from that address.

5. Explain Characteristics and issues related to stream communication.

Ans:

Characteristics

- **Message Sizes:** The application can choose how much data it writes to a stream or reads from it. It may deal with very small or very large sets of data. Inter-Process Communication 8 The underlying implementation of a TCP stream decides how much data to collect before transmitting it as one or more IP packets. On arrival, the data is handled to the application as requested. Applications can, if necessary, force data to be sent immediately.
- **Lost Messages:** The TCP protocol uses an acknowledgement scheme. As an example of a simple scheme, the sending end keeps a record of each IP packet sent and the receiving end acknowledges all the arrivals. If the sender does not receive an acknowledgement within a timeout, it re-transmits the message.

- **Flow Control:** The TCP protocol attempts to match the speed of the processes that read from and write to a stream. If the writer is too fast for the reader, then it is blocked until the reader has consumed sufficient data.
- **Message Duplication and Ordering:** Message identifiers are associated with each IP packet, which enables the recipient to detect and reject duplicates, or to reorder the messages that do not arrive in sender order.
- **Message Destinations:** A pair of communication processes establish a connection before they can communicate over a stream. Once a connection is established, the processes simply read from and write to the stream without needing the use of internet addresses and ports. Establishing a connection involves a connect request from the client to the server followed by an accept request from the server to the client before any communication can take place. This could be a considerable overhead for a single client-server request and reply.

Issues

- **Matching of Data Items:** Two communicating processes need to agree as to the contents of the data transmitted over a stream. For example, if one process writes an int followed by a double to a stream, then the reader at the other end must also read an int followed by a double. When a pair of processes do not cooperate correctly in their use of a stream, the reading process may experience errors when interpreting the data.
- **Blocking:** The data written to a stream is kept in a queue at the destination socket. When a process attempts to read data from an input channel, it will get data from the queue or it will block until data becomes available. During this step, the process that writes data to a stream may be blocked by the TCP flow mechanism.
- **Threads:** When a server accepts a connection, it generally creates a new thread in which to communicate with the new client. The advantage of using a separate thread for each client is that the server can block when waiting for input without delaying other clients.

6. Define marshalling and unmarshalling. Explain CORBA CDR with an example.

Ans:

- **Marshalling:** It is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message. The purpose of the data marshalling mechanism is to represent the caller's argument in a way that can be efficiently interpreted by a server program.
- **Unmarshalling:** It is the process of disassembling a collection of data on arrival to produce an equivalent collection of data items at the destination.

CORBA CDR:

CORBA's Common Data Representation is the external data representation defined with CORBA 2.0. CDR can represent all of the data types that can be used as arguments and return values in remote invocations in CORBA. These consist of 15 primitive types, which include short (16-bit), long (32-bit), unsigned short, unsigned long, float, double, char, boolean, octet, and any.
Example: Person struct with value: {'Smith', 'London', 1984}

index in sequence of bytes	←4 bytes→	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20–23	"on__"	
24–27	1984	*unsigned long*

7. Explain Java object serialization with an example.

Ans:

In Java RMI, both objects and primitive data values may be passed as arguments and results of method invocations. An object is an instance of a Java class. For example, the Java class equivalent to the Person struct defined in CORBA IDL might be:

```
public class Person implements Serializable {
        private String name;
        private String place;
        private int year;
    public Person(String aName, String aPlace, int aYear) {
            name = aName;
            place = aPlace;
            year = aYear;
    }
            // followed by methods for accessing the instance variables
}
```

The above class states that it implements the Serializable interface, which has no methods. Stating that a class implements the Serializable interface (which is provided in the java.io package) has the effect of allowing its instances to be serialized.

To serialize an object, its class information is written out, followed by the types and names of its instance variables. If the instance variables belong to new classes, then their class information must also be written out, followed by the types and names of their instance variables. This recursive procedure continues until the class information and types and names of the instance variables of all of the necessary classes have been written out. Each class is given a handle, and no class is written more than once to the stream of bytes (the handles being written instead where necessary).

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name	java.lang.String place	*number, type and name of instance variables*
1984	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers; h0 and h1 are handles.

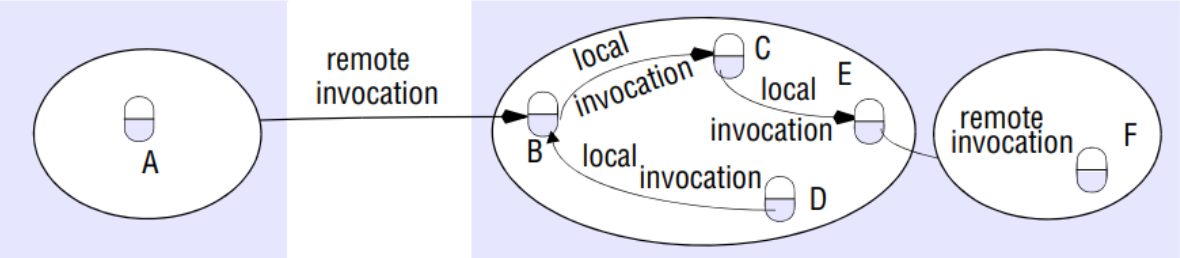
8. **Define Marshalling. Construct a marshalled form that represents a Organization with instance variable values :{ 'KLSGIT','BELGAUM', 1979, 590008} by using CORBACDR & Java Serialization.**

   Ans: Refer previous answers.

Chapter 4 - Distributed Object and RMI.

9. Explain communication between distributed objects by means of RMI.

Ans:



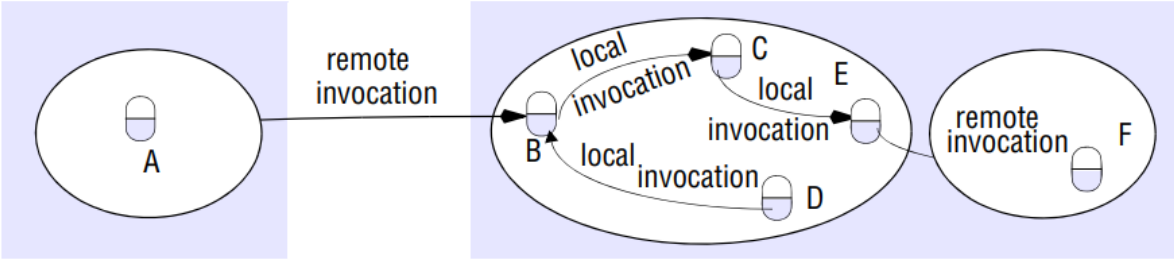
- The term distributed objects usually refer to software modules that are designed to work together but reside either in multiple computers connected via a network or in different processes inside the same computer.
- The state of an object consists of the values of its instance variables.
- Distributed object systems may adopt the client-server architecture. In this case, objects are managed by servers and their clients invoke their methods using Remote Method Invocation (RMI).
- In RMI, the client's request to invoke a method of an object is sent in a message to the server managing the object. The invocation is carried out by executing a method of the object at the server and the result is returned to the client in another message.
- To allow for chains of related invocation, objects in servers are allowed to become clients in other servers.
- **Transport Layer:** Connects client to the server.
- **Stub:** Representation (proxy) of the remote object at the client.
- **Skeleton:** Representation (proxy) of the remote object at the client.
- **RRL (Remote Reference Layer):** Manages references made by the client to the remote object.

Working of RMI Application

- When a client makes a request to the remote object, it is received by stub and eventually, the request is passed to RRL.
- When client-side RRL receives the request and it calls the method invoke() of the object remoteRef.
- It passes the request to RRL on the server-side. The RRL on the server-side passes the request to the skeleton which then finally executes the required object on the server.
- The result is passed all the way back to the client.

10. Explain remote and local invocation with the neat diagrams.

Ans:



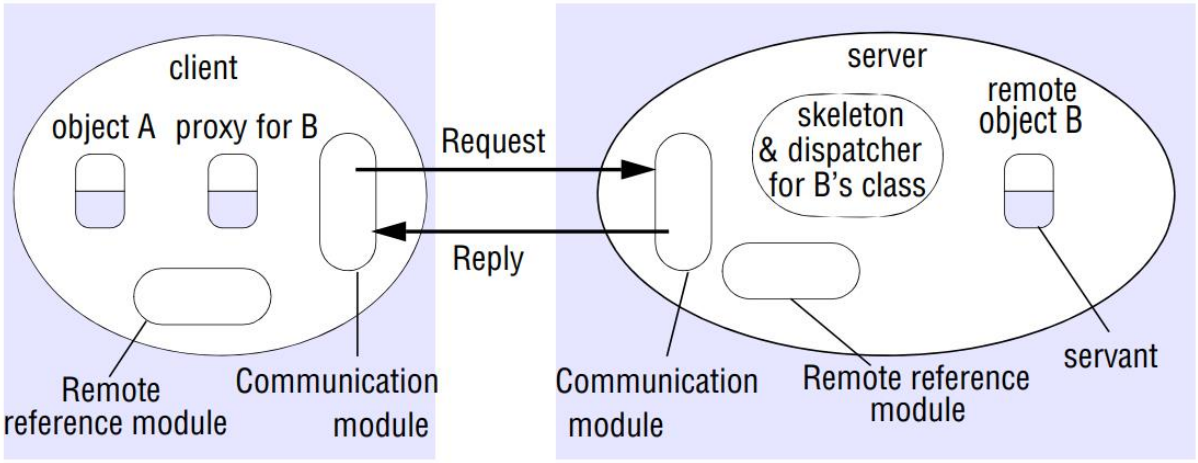
The fact that objects are accessed only via their methods gives rise to another advantage of heterogeneous systems, that different data formats may be used at different sites – these formats will be unnoticed by clients that use RMI to access the methods of the objects.

The distributed object model:

- This section discusses extensions to the object model to make it applicable to distributed objects.
- Each process contains a collection of objects, some of which can receive both local and remote invocations, whereas the other objects can receive only local invocations, as shown in Figure.
- Method invocations between objects in different processes, whether in the same computer or not, are known as remote method invocations.
- Method invocations between objects in the same process are local method invocations.
- We refer to objects that can receive remote invocations as remote objects. In Figure, the objects B and F are remote objects.
- All objects can receive local invocations, although they can receive them only from other objects that hold references to them. For example, object C must have a reference to object E so that it can invoke one of its methods. The following two fundamental concepts are at the heart of the distributed object model:
    - Remote object references: Other objects can invoke the methods of a remote object if they have access to its remote object reference. For example, a remote object reference for B in Figure must be available to A.
    - Remote interfaces: Every remote object has a remote interface that specifies which of its methods can be invoked remotely. For example, the objects B and F in Figure must have remote interfaces.

11. With a neat diagram explain the role of Proxy & Skeleton in RMI.

Ans:



RMI uses proxy and skeleton objects for communication with the remote object. A remote object is an object whose method can be invoked from another JVM.

Proxy

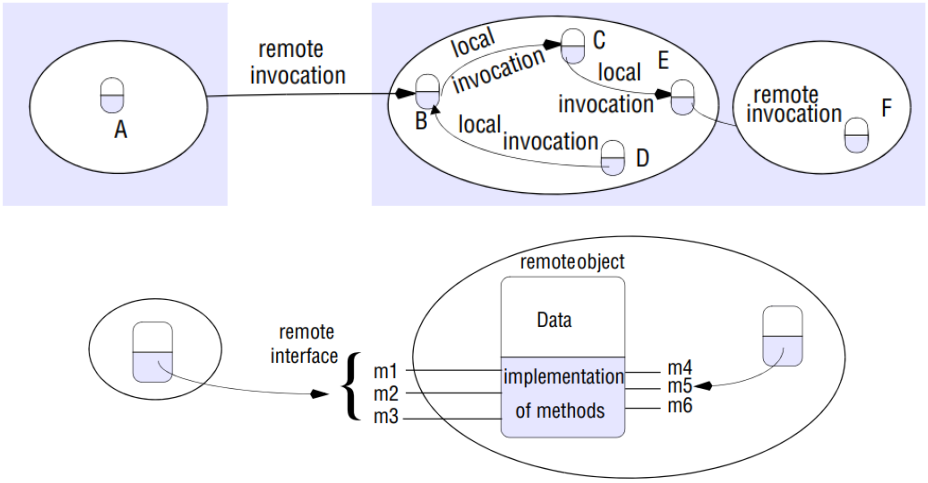
- The role of the proxy is to make remote method invocation transparent to clients by behaving like a local object to the invoker; but instead of executing an invocation, it forwards it in a message to the remote object.
- The proxy acts as a gateway for the client-side. All the outgoing requests are routed through it.
- When the caller invokes method on the stub object, it does the following tasks:
  - It initiates a connection with the remote Virtual Machine (JVM).
  - It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM).
  - It waits for the result.
  - It reads (unmarshals) the return value or exception and finally, it returns the value to the caller.
- There is one proxy for each remote object for which a process holds a remote object reference.

Skeleton

- The skeleton acts as a gateway for the server-side object.
- All the incoming requests are routed through it.
- When the skeleton receives the incoming request, it does the following tasks:
  - It reads (unmarshals) the parameter for the remote method.
  - It invokes the method on the actual remote object.
  - It writes and transmits (marshals) the result to the caller.

12. Explain the fundamental concepts of the distributed object model.

Ans:



- Each process contains a collection of objects, some of which can receive both local and remote invocations, whereas the other objects can receive only local invocations.
- Method invocations between objects in different processes, whether in the same computer or not, are known as remote method invocations.
- Method invocations between objects in the same process of the computer are local method invocations and objects that can receive remote invocations are called remote objects.
- There are two fundamental concepts in distributed object model:

i) **Remote Object References**: A remote object reference is an identifier that can be used throughout a distributed system to refer to a particular unique remote object. Other objects can invoke the methods of a remote object if they have access to its remote object reference. For example, a remote object reference for B in the figure must be available to A.
   - The remote object receiving a remote method invocation is specified by the invoker as a remote object reference.
   - Remote object references may be passed as an argument.

ii) **Remote Interfaces**:
   - The class of a remote object implements the methods of its remote interface.
   - It specifies which methods can be invoked remotely.
   - Objects in other processes can invoke only the methods that belong to its remote interface.
   - Remote interfaces, like all interfaces, do not have constructors.
   - The CORBA system provides an Interface Definition Language (IDL), which is used for defining remote interfaces.

13. Discuss RMI invocation semantics and tabulate failure handling mechanism for each.

Ans:

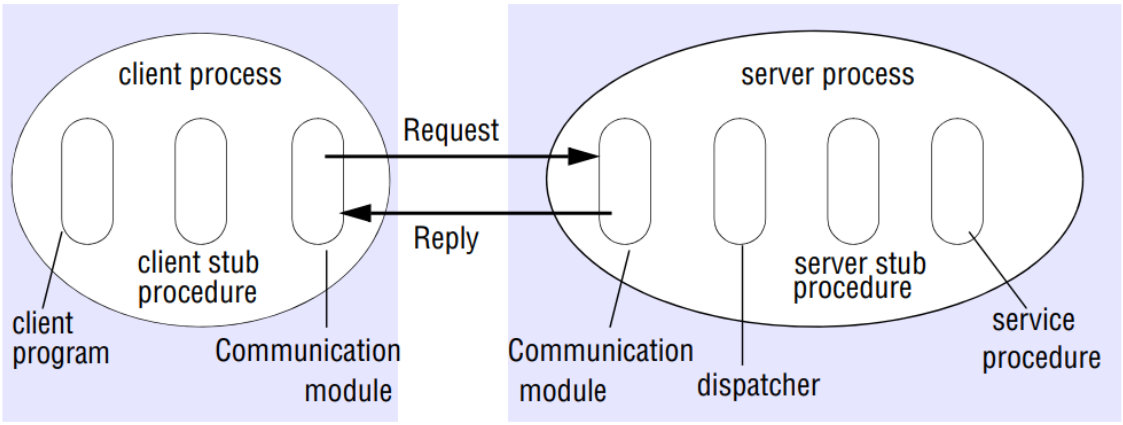
The RMI invocation semantics are as follows:

- **Maybe Semantics**
  - With maybe semantics, the remote procedure call may be executed once or not at all.
  - Maybe semantics arises when no fault-tolerance measures are applied and can suffer from the following types of failures:
    - Omission failures if the request or result message is lost.
    - Crash failures when the server containing the remote operation fails.
  - If the result message has not been received after a timeout and there are no retries, it is uncertain whether the procedure has been executed.
- **At-least-once Semantics**
  - With at-least-once semantics, the invoker receives wither a result, in which case the invoker knows that the procedure was executed at least once, or an exception informing it that no result was received.
  - At-least-once semantics can be achieved by the retransmission of request messages.
  - At-lease-once semantics can suffer from the following types of failures:
    - Crash failures when the server containing the remote procedure fails.
    - Arbitrary failures - in cases when the request message is retransmitted, the remote server may receive it and execute the procedure more than once, possibly causing wrong values to be stored and returned.
- **At-most-once Semantics**
  - With at-most-once semantics, the caller receives either a result, in which case the caller knows that the procedure was executed exactly once, or an exception informing it that no result was received, in which case the procedure will have been executed either once or not at all.
- **Failure handling mechanism**

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

14. Define RPC and with neat diagram explain its implementation.

Ans:



The Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details.

An RPC is analogous to a function call. Like a function call, when RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure.

Implementation

- The software components required to implement RPC are shown in the figure.
- The client calls the client stub. The call is a local procedure call, with parameters pushed onto the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called Marshalling.
- The client's local operating system sends the message from the client machine to the server machine.
- The local operating system on the server machine passes the incoming packets to the server stub.
- The server stub unpacks the parameters from the message. Unpacking the parameters is called Unmarshalling.
- Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

15. Explain HTTP request and reply message format.

Ans:

An HTTP request message typically includes the following elements:

method	*URL or pathname*	*HTTP version*	*headers*	*message body*
GET	http://www.dcs.qmul.ac.uk/index.html	HTTP/ 1.1		

- *Method:* This specifies the request type, such as GET, POST, PUT, DELETE, etc.
- *URI (Uniform Resource Identifier):* This specifies the resource being requested, such as a web page or image.
- *HTTP version:* This specifies the version of HTTP being used, such as HTTP/1.1.
- *Headers:* This includes additional information about the request, such as the client's preferred language or the type of data being sent in the request body.
- *Body:* This includes any data being sent with the request, such as form data or a JSON payload.

An HTTP response message typically includes the following elements:

HTTP version	*status code*	*reason*	*headers*	*message body*
HTTP/1.1	200	OK		resource data

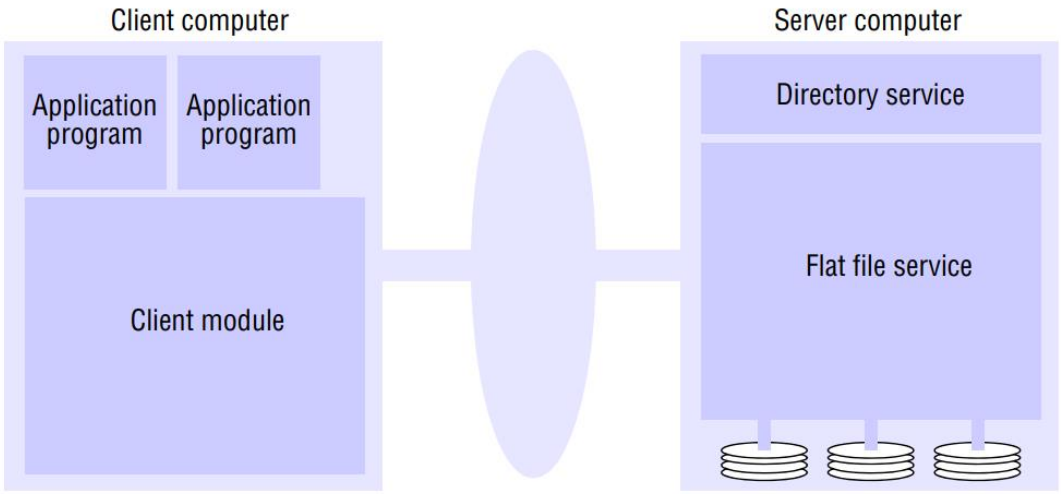
- *HTTP version:* This specifies the version of HTTP being used, such as HTTP/1.1.
- *Status code:* This is a 3-digit numerical code indicating the outcome of the request. 200 OK, 404 Not Found, 403 forbidden, 401 Unauthorized are common status codes.
- *Reason phrase:* This is a brief, human-readable description of the status code.
- *Headers:* This includes additional information about the response, such as the type of data being sent in the response body or the server's software version.
- *Body:* This includes any data being sent with the response, such as the requested web page or an error message.

Unit – 3

Chapter 5 - Distributed File System.

1. **Discuss model architecture of distributed file system and its components.**

Ans:



File service architecture is an architecture that offers a clear separation of the main concerns in providing access to files is obtained by structuring the file service as three components:

Flat File Service:

- FFS is concerned with the implementation of operations on the contents of the file.
- Unique File Identifiers (UFIDs) are used to refer to files in all requests for flat-file service operations.
- UFIDs are long sequences of bits chosen so that each file has a UFID that is unique among all of the files in a distributed system.
- When the flat file service receives a request to create a file, it generates a new UFID for it and returns the UFID to the requester.

Directory Service:

- The Directory Service provides a mapping between text names for the files and their UFIDs.
- Clients may obtain the UFID of a file by quoting its text name to the directory service.
- The directory service provides the functions needed to generate directories, to add new files to directories, and obtain UFIDs from directories.

Client Module:

- A Client Module runs in each client computer, integrating and extending the operations of the flat file service and the directory service under a single application programming interface that is available to user-level programs in client computers.
- It holds information about the network locations of flat-file and directory server processes.
- It helps to achieve better performance through the implementation of a cache of recently used file blocks at the client.

2. **With a neat diagram explain the components of file service architecture in brief w. r. t. following;**
   - **i) Flat File Service**
   - **ii) Directory Service**
   - **iii) Client Module**

Ans: Refer previous question.

3. **List out file system modules.**

Ans:

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	performs disk I/O and buffering

4. **Sketch the file attributes and record structure.**

Ans:

File length
Creation timestamp
Read timestamp
Write timestamp
Attribute timestamp
Reference count
Owner
File type
Access control list

5. List out the transparencies in file system.

Ans:

The file service is usually the most heavily loaded service in an intranet, so its functionality and performance are critical. The design must balance the flexibility and scalability that derive from transparency against software complexity and performance. The following forms of transparency are partially or wholly addressed by current file services:

- **Access transparency:** Client programs should be unaware of the distribution of files. A single set of operations is provided for access to local and remote files. Programs written to operate on local files are able to access remote files without modification.
- **Location transparency:** Client programs should see a uniform file name space. Files or groups of files may be relocated without changing their pathnames, and user programs see the same name space wherever they are executed.
- **Mobility transparency:** Neither client programs nor system administration tables in client nodes need to be changed when files are moved. This allows file mobility – files or, more commonly, sets or volumes of files may be moved, either by system administrators or automatically.
- **Performance transparency:** Client programs should continue to perform satisfactorily while the load on the service varies within a specified range.
- **Scaling transparency:** The service can be expanded by incremental growth to deal with a wide range of loads and network sizes.

6. List the directory service operation.

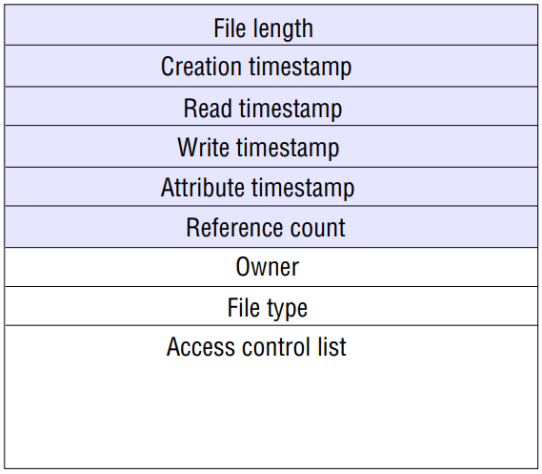
Ans:

Lookup(Dir, Name) → *FileId* — throws *NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, FileId) — throws *NameDuplicate*	If *Name* is not in the directory, adds *(Name, File)* to the directory and updates the file's attribute record. If *Name* is already in the directory, throws an exception.
UnName(Dir, Name) — throws *NotFound*	If *Name* is in the directory, removes the entry containing *Name* from the directory. If *Name* is not in the directory, throws an exception.
GetNames(Dir, Pattern) → *NameSeq*	Returns all the text names in the directory that match the regular expression *Pattern*.

7. Describe the characteristics of file system.

Ans:

- File systems are responsible for the organization, storage, retrieval, naming, sharing and protection of files.
- They provide a programming interface that characterizes the file abstraction, freeing programmers from concern with the details of storage allocation and layout.
- Files are stored on disks or other non-volatile storage media.



- Files contain both data and attributes.
- The data consist of a sequence of data items (typically 8-bit bytes), accessible by operations to read and write any portion of the sequence.
- The attributes are held as a single record containing information such as the length of the file, timestamps, file type, owner's identity and access control lists.
- A typical attribute record structure is illustrated in Figure. The shaded attributes are managed by the file system and are not normally updatable by user programs.
- File systems are designed to store and manage large numbers of files, with facilities for creating, naming and deleting files.
- The naming of files is supported by the use of directories.
- A directory is a file, often of a special type, that provides a mapping from text names to internal file identifiers.
- Directories may include the names of other directories, leading to the familiar hierarchic file-naming scheme and the multi-part pathnames for files used in UNIX and other operating systems.
- File systems also take responsibility for the control of access to files, restricting access to files according to users' authorizations and the type of access requested (reading, updating, executing and so on).

8. Discuss the distributed file system design requirements.

Ans:

- **Transparency:** Some aspects of a Distributed system are hidden from the user.
    - **Access:** Client programs can be unaware of the distribution of files. The same set of operations is provided for access to remote as well as local files. Location: The client program should see a uniform namespace.
    - **Mobility:** Client programs need not change their tables when files are moved to any other location.
    - **Performance:** The client program should continue to perform satisfactorily while the load on the service varies.
    - **Scaling:** The service can be expanded to deal with a wide range of load and network sizes.
    - **Location:** Client programs should see a uniform file name space. Files or groups of files may be relocated without changing their pathnames, and user programs see the same name space wherever they are executed.
- **Concurrent file updates:** Changes to the file by one client should not interfere with the operation of other clients simultaneously accessing or changing the same file.
- **File replication:** A file may be represented by several copies of its contents at different locations. It has the following benefits:
    i. Load balancing to enhance the scalability of the service.
    ii. Enhances the fault tolerance.
- **Hardware and OS heterogeneity:** The service interfaces should be defined so that client and server software can be implemented for different operating systems and computers.
- **Fault tolerance:** To cope with transient communication failures, the design can be based on at-most-once invocation semantics.
- **Consistency:** Maintaining the consistency between multiple copies of files.
- **Security:** In distributed file systems, there is a need to authenticate client requests so that access control at the server is based on correct user identities and to protect the contents of the request and reply messages.
- **Efficiency:** A distributed file service should offer facilities that are of at least the same power and generality as those found in conventional file systems and should achieve a comparable level of performance.

Chapter 6 - Security in distributed systems.

1. Write the steps of RSA Algorithm. Illustrate with an example given P=3 & Q=11.

Ans:

RSA Algorithm

1. Choose two large prime numbers P and Q, where each is greater than 10^{100}. Calculate N = P * Q and φ(n) = (P-1) * (Q-1).
2. Choose a number e that is relatively prime to φ(n) and 1 < e < φ(n).
3. Find d by solving e * d = 1 mod φ(n), where e * d is the smallest element divisible by d in the series Z+1, 2Z+1, 3Z+1...
4. Private Key: (d, N), Public Key: (e, N)
5. Plaintext M is converted to ciphertext C using the formula: $C = M^e \bmod N$
6. Ciphertext C is converted to plaintext M using the formula: $M = C^d \bmod N$

RSA Algorithm applied for P = 3, Q = 11 and M = 10.

1. P = 3, Q = 11 -> N = 3 * 11 = 33 and φ(n) = (3-1) * (11-1) = 20
2. Choose e = 7, since 7 is relatively prime to 20 (1 < 7 < 20).
3. Solve e * d = 1 mod 20 to find d. The modular inverse of 7 (mod 20) is 15, so d = 15.
4. Private Key: (15, 33), Public Key: (7, 33)
5. To encrypt plaintext M = 10, calculate $C = 10^7 \bmod 33 = 27$.
6. To decrypt ciphertext C = 27, calculate $M = 27^{15} \bmod 33 = 10$.

2. Analyze the following uses of Cryptography with suitable scenarios.

I. Secrecy and integrity

II. Authentication

Ans:

i) Secrecy and Integrity

- Cryptography is used to maintain the secrecy and integrity of information whenever it is exposed to potential attacks.
- It maintains the secrecy of the encrypted message as long as the decryption key is not compromised.

- It also maintains the integrity of the encrypted information, provided that some redundant information such as a checksum is included and checked.

  Scenario: Secret communication with a shared secret key: Alice wishes to send some information secretly to Bob. Alice and Bob share a secret key KAB.

- Alice uses KAB and an agreed encryption function E(KAB, M) to encrypt and send any number of messages {Mi}KAB to Bob.
- Bob decrypts the encrypted messages using the corresponding decryption function D(KAB, M).

ii) Authentication

- Cryptography is used in support of mechanisms for authenticating communication between pairs of principals.
- Key is known only to two parties.

  Scenario: Bob has a public/private key pair <KBpub, KBpriv>

- Alice obtains a certificate that was signed by a trusted authority stating Bob's public key Kbpub.
- Alice creates a new shared key KAB, encrypts it using Kbpub using a public-key algorithm, and sends the result to Bob.
- Bob uses the corresponding private key KBpriv to decrypt it.

3. Discuss asymmetric (public/private key pair-based) cryptography technique and how it can be used in supporting security in distributed systems.

Ans:

When a public/private key pair is used, one-way functions are exploited in another way. The feasibility of a public-key scheme was proposed as a cryptographic method that eliminates the need for trust between communicating parties. The basis for all public-key schemes is the existence of a trap-door function. A trap-door function is a one-way function with a secret exit- it is easy to compute in one direction but infeasible to compute its reverse unless the secret is known.

The pair of keys needed for asymmetric algorithms are derived from a common root. The derivation of the pair of keys from the root is a one-way function. In the case of the RSA algorithm, the large numbers are multiplied together, this computation takes only a few seconds, even for very large primes used. The resulting product, N, is computationally infeasible to derive the original multiplicands.

One of the pair of keys is used for encryption. For RSA the encryption procedure obscures the plaintext by treating each block as a binary number and raising it to the power of key, modulo N. The resulting number is the corresponding ciphertext block. The size of N and at least one of the pair of keys is much larger than the safe key size for symmetric keys to ensure that N is not factorizable. For this reason, the potential attacks on RSA are small, its resistance to attacks depends on the infeasibility of factorizing N.

4. What is a distributed denial-of-service attack and how does it work?

Ans:

A distributed denial-of-service (DDoS) attack is a type of cyber-attack in which multiple devices, often compromised with malware, are used to flood a single target with large amounts of traffic in an attempt to overload its servers and prevent legitimate users from accessing it. The goal of a DDoS attack is to disrupt the normal traffic of a targeted server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic.

Here are the general steps that may be involved in a DDoS attack:

[1]. The attacker identifies a target, which can be a website, server, or network.

[2]. The attacker infects a large number of devices, such as personal computers or IoT devices, with malware that allows the attacker to remotely control them. These infected devices are referred to as "bots" or "zombies."

[3]. The attacker uses the bots to launch a coordinated flood of requests to the target, overwhelming its servers and preventing legitimate traffic from getting through.

[4]. The attack traffic may come from a single source or multiple sources, making it difficult to block or filter.

[5]. The target's servers and network infrastructure become overwhelmed and unable to respond to legitimate requests, causing the targeted website or service to become unavailable to users.

[6]. The attack continues until the attacker chooses to stop it or the target takes measures to block or mitigate the attack traffic.

It's important to note that there are different types of DDoS attacks and the exact method used can vary depending on the type of attack and the target. Additionally, the attackers are using new techniques like using botnets, amplification, spoofing, etc to make their attack more sophisticated.

5. What is the goal of security? List the three broad classes of security threats?

Ans:

The main goal of security is to restrict access to information and resources to just those principals that are authorized to have access. Security threats fall into three broad classes:

- **Leakage:** Refers to the acquisition of information by unauthorized recipients.
- **Tampering**: Refers to the unauthorized alteration of information.
- **Vandalism**: Refers to interference with the proper operation of a system without gain to the perpetrator.

6. What is cryptography? What is the use of it?

Ans:

Cryptography is the practice of secure communication in the presence of third parties. It is a technique for secure communication that uses mathematical algorithms to encrypt and decrypt messages, ensuring that only the intended recipient can read the message. Cryptography plays a critical role in distributed computing, as it allows for secure communication between nodes in a distributed system.

- **Secrecy and integrity**: Cryptography can be used to protect the secrecy and integrity of data in a distributed system. One common technique for achieving this is to use symmetric encryption to encrypt the data, and then use a message authentication code (MAC) to ensure that the data has not been tampered with in transit. The recipient can then use the same symmetric key to decrypt the data and check the MAC to verify the integrity of the data.
- **Authentication**: Cryptography can be used to authenticate the identity of parties in a distributed system. One common technique for achieving this is to use public key infrastructure (PKI) which involves the use of a public key to encrypt messages and a private key to decrypt them. By encrypting a message with a public key, the sender can ensure that only the holder of the corresponding private key can read it, thus providing a way to authenticate the identity of the recipient.
- **Digital signatures**: Cryptography can be used to provide digital signatures, which are a way of verifying the authenticity of a message or document. A digital signature is created by encrypting a message with the sender's private key, and can be verified by decrypting it with the corresponding public key. This ensures that the message was sent by the holder of the private key, and that the message has not been tampered with in transit. Digital signatures are commonly used in electronic commerce and other applications to provide non-repudiation.

7. Write a note on digital signature.

Ans:

A digital signature is a method of verifying the authenticity and integrity of a digital document or message. It is created by encrypting a message or document using the private key of the sender, and can be verified by decrypting it using the corresponding public key. A digital signature provides several benefits:

1. **Authentication**: A digital signature verifies that the message or document was sent by the holder of the private key, and not by an imposter.
2. **Integrity:** A digital signature ensures that the message or document has not been tampered with in transit, as any changes made to the message will result in a different digital signature.
3. **Non-repudiation**: A digital signature provides evidence that the sender cannot later deny having sent the message or document.

Digital signatures are commonly used in a variety of applications such as electronic commerce, email, and software distribution.

There are two main types of digital signatures:

1. **Symmetric digital signature**: One key is used for signing and verification.
2. **Asymmetric digital signature**: Public key is used for signing and private key is used for verification.

Digital signature schemes such as RSA, DSA, ECDSA etc are widely used for digital signature.

Unit – 4

Chapter – 7: Time and Global States

1. **Define following terms**
   - **Physical clock**
   - **Clock skew and clock drift**
   - **Coordinated Universal Time**

Ans:

a. **Physical clock**: A physical clock is a hardware device that keeps track of time. Physical clocks are used in distributed systems to provide a synchronized time reference for all nodes in the system. They are typically based on quartz crystals or atomic oscillators and have a high degree of accuracy and stability.

b. **Clock skew**: Clock skew refers to the difference in time between two clocks that are supposed to be synchronized. In a distributed system, clock skew can occur due to differences in the speed at which different clocks are running or due to delays in the time it takes for time updates to propagate through the system. Clock skew can have a significant impact on the performance and correctness of distributed algorithms.

c. **Clock drift**: Clock drift refers to the gradual change in the rate at which a clock is running. Clock drift can occur due to changes in temperature, humidity, power supply voltage, or other factors. In a distributed system, clock drift can cause the clocks on different nodes to drift out of synchronization over time.

d. **Coordinated Universal Time (UTC):** Coordinated Universal Time is a standardized time reference that is used in many distributed systems. UTC is based on the International Atomic Time (TAI) and is adjusted for leap seconds to keep it close to the rotation of the Earth. UTC is used as a reference time in many applications such as navigation, telecommunications, and computer networks.

2. **Explain different modes of synchronizing a physical clocks.**

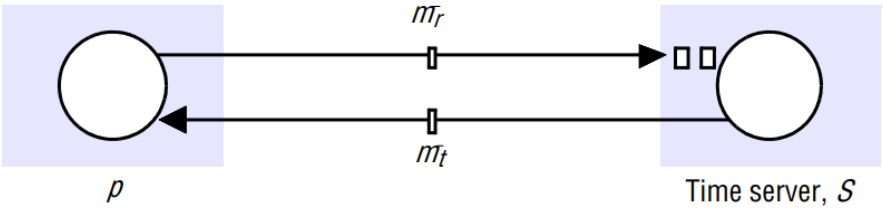
Ans:

1. **External synchronization**: External synchronization refers to the use of an external time reference to synchronize the clocks in a distributed system. The external time reference can be a GPS receiver, an atomic clock, or a time server connected to the Internet. The nodes in the distributed system use this external time reference to adjust their clocks and maintain synchronization. External synchronization is typically used in systems where accurate time synchronization is critical, such as financial transactions or scientific experiments.
2. **Internal synchronization**: Internal synchronization refers to the use of internal algorithms and protocols to synchronize the clocks in a distributed system. The internal algorithms and protocols can be based on message passing or on the measurement of network delays. Internal synchronization is typically used in systems where the network is unreliable or the external time reference is unavailable. The internal algorithms and protocols can be less accurate than external synchronization, but they can still provide good synchronization for most distributed systems.

It's worth noting that both external and internal synchronization methods can be used together to achieve better synchronization. For example, a distributed system may use an external time source to provide a coarse time synchronization, and then use internal algorithms to fine-tune the synchronization.

3. Explain Cristian's method for synchronizing clocks.

Ans:



- A device that receives signals from a source of UTC, to synchronize computers externally. Upon request, the server process S supplies the time according to its clock, as shown in Figure.
- The method achieves synchronization only if the observed round-trip times between client and server are sufficiently short compared with the required accuracy.
- A process p requests the time in a message m_r, and receives the time value t in a message m_t.
- Process p records the total round-trip time T_{round} taken to send the request mr and receive the reply m_t. It can measure this time with reasonable accuracy if its rate of clock drift is small.
- A simple estimate of the time to which ***p*** should set its clock is $t + T_{round} / 2$, which assumes that the elapsed time is split equally before and after ***S*** placed t in m_t.
- This is normally a reasonably accurate assumption, unless the two messages are transmitted over different networks. If the value of the minimum transmission time min is known or can be conservatively estimated, then we can determine the accuracy of this result as follows.
- The earliest point at which S could have placed the time in m_t was ***min*** after ***p*** dispatched m_r.
- The latest point at which it could have done this was ***min*** before m_t arrived at p.
- The time by ***S***'s clock when the reply message arrives is therefore in the range $[t + min, t + T_{round} - min]$.
- The width of this range is $T_{round} - 2\ min$, so the accuracy is $\pm(T_{round} / 2 - min)$.
- Variability can be dealt with to some extent by making several requests to ***S*** and taking the minimum value of T_{round} to give the most accurate estimate.
- The greater the accuracy required, the smaller the probability of achieving it. This is because the most accurate results are those in which both messages are transmitted in a time close to ***min –*** an unlikely event in a busy network.

4. Explain Berkeley algorithm for internal synchronization.

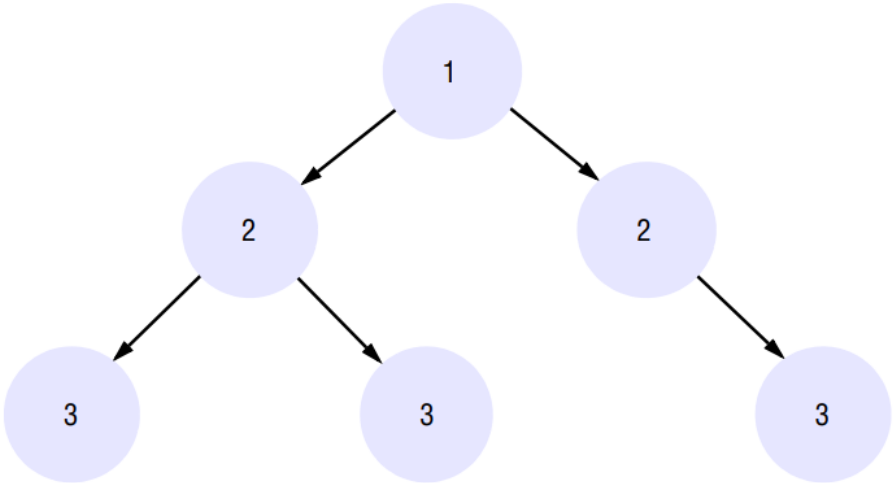
Ans:

- A coordinator computer is chosen to act as the master.
- This computer periodically polls the other computers whose clocks are to be synchronized, called slaves.
- The slaves send back their clock values to it. The master estimates their local clock times by observing the round-trip times (similarly to Cristian's technique), and it averages the values obtained (including its own clock's reading).
- The balance of probabilities is that this average cancel out the individual clocks' tendencies to run fast or slow.
- The accuracy of the protocol depends upon a nominal maximum round-trip time between the master and the slaves. The master eliminates any occasional readings associated with larger times than this maximum.
- Instead of sending the updated current time back to the other computers – which would introduce further uncertainty due to the message transmission time – the master sends the amount by which each individual slave's clock requires adjustment. This can be a positive or negative value.
- The Berkeley algorithm eliminates readings from faulty clocks. Such clocks could have a significant adverse effect if an ordinary average was taken so instead the master takes a *fault-tolerant average*. That is, a subset is chosen of clocks that do not differ from one another by more than a specified amount, and the average is taken of readings from only these clocks.

5. With the neat diagram, explain the concept of synchronization subnet in an NTP implementation.

Ans:

An example synchronization subnet in an NTP implementation



- The NTP service is provided by a network of servers located across the Internet. Primary servers are connected directly to a time source such as a radio clock receiving UTC; secondary servers are synchronized, ultimately, with primary servers.
- The servers are connected in a logical hierarchy called a synchronization subnet whose levels are called strata.
- Primary servers occupy stratum 1: they are at the root. Stratum 2 servers are secondary servers that are synchronized directly with the primary servers; stratum 3 servers are synchronized with stratum 2 servers, and so on.
- The lowest-level (leaf) servers execute in users' workstations The clocks belonging to servers with high stratum numbers are liable to be less accurate than those with low stratum numbers, because errors are introduced at each level of synchronization.
- NTP also takes into account the total message round-trip delays to the root in assessing the quality of timekeeping data held by a particular server. The synchronization subnet can reconfigure as servers become unreachable or failures occur.
- If, for example, a primary server's UTC source fails, then it can become a stratum 2 secondary server.
- If a secondary server's normal source of synchronization fails or becomes unreachable, then it may synchronize with another server.
- NTP servers synchronize with one another in one of three modes: multicast, procedure-call and symmetric mode. Multicast mode is intended for use on a high-speed LAN.
- One or more servers periodically multicasts the time to the servers running in other computers connected by the LAN, which set their clocks assuming a small delay.
- This mode can achieve only relatively low accuracies, but ones that nonetheless are considered sufficient for many purposes.

6. Discuss different modes of NTP server synchronization.

Ans:

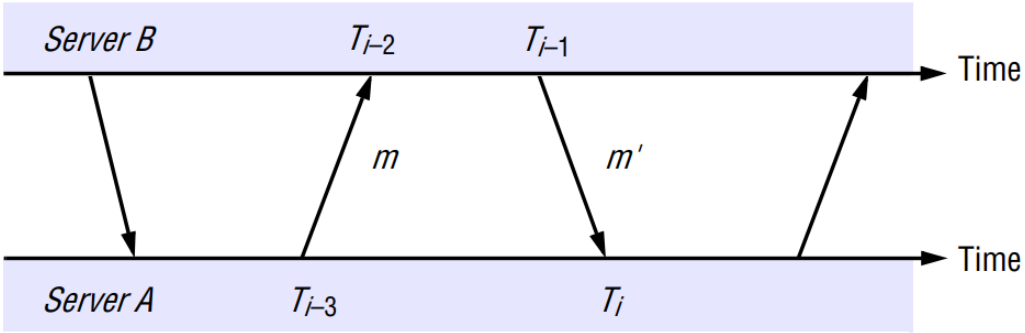
In Network Time Protocol (NTP) server synchronization, there are several different modes that can be used to synchronize the clocks of servers in a distributed system. These modes include:

- ***Multicast mode*** is intended for use on a high-speed LAN. One or more servers periodically multicasts the time to the servers running in other computers connected by the LAN, which set their clocks assuming a small delay. This mode can achieve only relatively low accuracies, but ones that nonetheless are considered sufficient for many purposes.
- ***Procedure-call mode*** is similar to the operation of Cristian's algorithm. In this mode, one server accepts requests from other computers, which it processes by replying with its timestamp (current clock reading). This mode is suitable where higher accuracies are required than can be achieved with multicast, or where multicast is not supported in hardware. For example, file servers on the same or a neighbouring LAN that need to keep accurate timing information for file accesses could contact a local server in procedure-call mode.
- ***Symmetric mode*** is intended for use by the servers that supply time information in LANs and by the higher levels (lower strata) of the synchronization subnet, where the highest accuracies are to be achieved. A pair of servers operating in symmetric mode exchange messages bearing

timing information. Timing data are retained as part of an association between the servers that is maintained in order to improve the accuracy of their synchronization over time.

7. Explain with the neat diagram how messages are exchanged between a pair of NTP peers.

Ans:



- In all modes, messages are delivered unreliably, using the standard UDP Internet transport protocol.
- In procedure-call mode and symmetric mode, processes exchange pairs of messages. Each message bears timestamps of recent message events:
  - The local times when the previous NTP message between the pair was sent and received.
  - The local time when the current message was transmitted.
- The recipient of the NTP message notes the local time when it receives the message. The four times $\mathbf{T_{i-3}}$, $\mathbf{T_{i-2}}$, $\mathbf{T_{i-1}}$ and $\mathbf{T_i}$ are shown in figure for the messages **m** and **m'** sent between servers A and B.
- In symmetric mode there can be a nonnegligible delay between the arrival of one message and the dispatch of the next.
- Messages may be lost, but the three timestamps carried by each message are nonetheless valid.
- For each pair of messages sent between two servers the NTP calculates an offset $\boldsymbol{o_i}$, which is an estimate of the actual offset between the two clocks, and a delay $\boldsymbol{d_i}$, which is the total transmission time for the two messages. If the true offset of the clock at ***B*** relative to that at ***A*** is ***o***, and if the actual transmission times for ***m*** and ***m'*** are ***t*** and ***t'***, respectively, then we have:

$$T_{i-2} = T_{i-3} + t + o \text{ and } T_i = T_{i-1} + t' - o$$

This leads to:

$$d_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}$$

and:

$$o = o_i + (t' - t)/2, \text{ where } o_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2$$

Using the fact that $t, t' \geq 0$, it can be shown that $o_i - d_i/2 \leq o \leq o_i + d_i/2$. Thus o_i is an estimate of the offset, and d_i is a measure of the accuracy of this estimate.

Chapter – 8: Coordination and Agreement

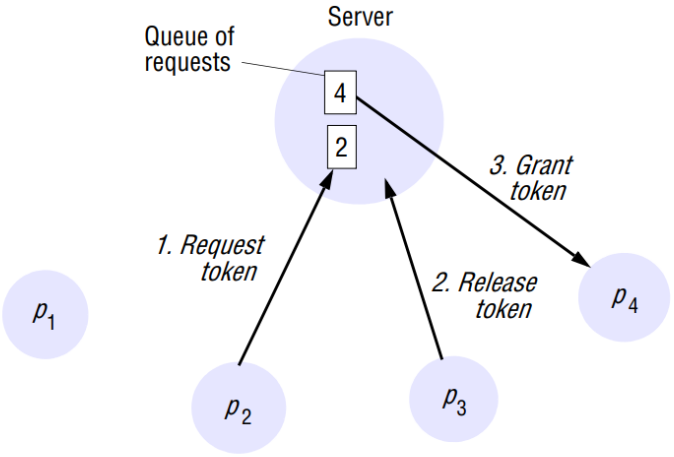
Note: Essential requirements for mutual exclusion are as follows:

- ME1: (safety) At most one process may execute in the critical section (CS) at a time.
- ME2: (liveness) Requests to enter and exit the critical section eventually succeed.
- ME3: (→ ordering) If one request to enter the CS happened-before another, then entry to the CS is granted in that order.

1. Explain with a neat diagram Central Server Algorithm.

Ans:

Server managing a mutual exclusion token for a set of processes



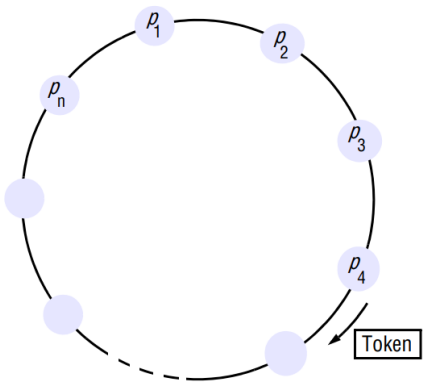
- The simplest way to achieve mutual exclusion is to employ a server that grants permission to enter the critical section. Figure shows the use of this server.
- To enter a critical section, a process sends a request message to the server and awaits a reply from it. The reply constitutes a token signifying permission to enter the critical section.
- If no other process has the token at the time of the request, then the server replies immediately, granting the token.
- If the token is currently held by another process, then the server does not reply, but queues the request.
- When a process exits the critical section, it sends a message to the server, giving it back the token.
- If the queue of waiting processes is not empty, then the server chooses the oldest entry in the queue, removes it and replies to the corresponding process.
- The chosen process then holds the token. In the figure, we show a situation in which p2 's request has been appended to the queue, which already contained p4 's request. p3 exits the critical section, and the server removes p4 's entry and grants permission to enter to p4 by replying to it. Process p1 does not currently require entry to the critical section.
- Given our assumption that no failures occur, it is easy to see that the safety and liveness conditions are met by this algorithm.
- The reader should verify, however, that the algorithm does not satisfy property ME3.

- We now evaluate the performance of this algorithm. Entering the critical section – even when no process currently occupies it – takes two messages (a request followed by a grant) and delays the requesting process by the time required for this round-trip.
- Exiting the critical section takes one release message. Assuming asynchronous message passing, this does not delay the exiting process.

2. With neat diagram explain Ring based Algorithm w.r.t. mutual exclusion.

Ans:

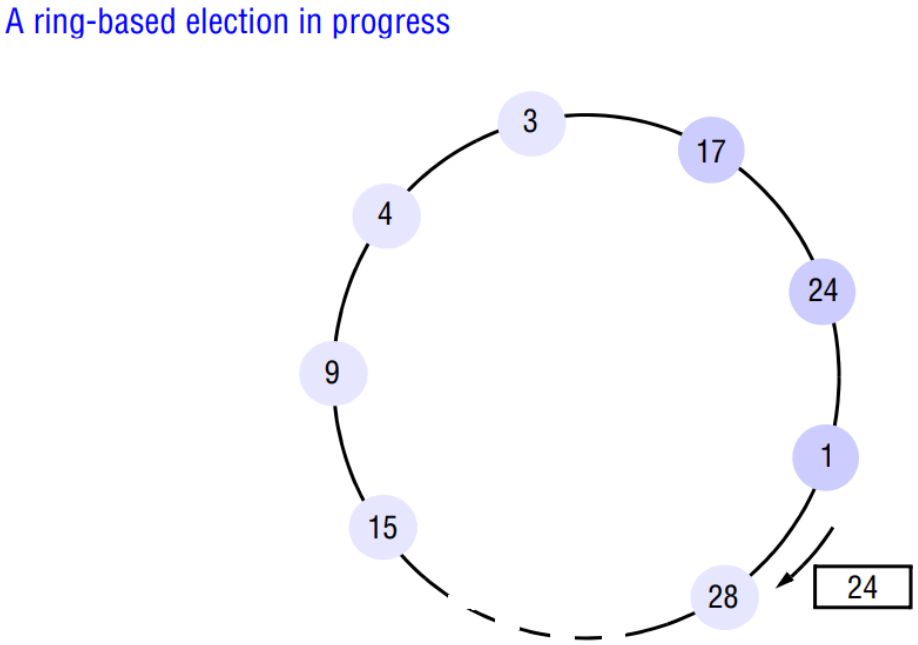
A ring of processes transferring a mutual exclusion token



- One of the simplest ways to arrange mutual exclusion between the ***N*** processes without requiring an additional process is to arrange them in a logical ring.
- This requires only that each process p_i has a communication channel to the next process in the ring, $p_{(i+1) \bmod N}$.
- The idea is that exclusion is conferred by obtaining a token in the form of a message passed from process to process in a single direction clockwise, say – around the ring. The ring topology may be unrelated to the physical interconnections between the underlying computers.
- If a process does not require to enter the critical section when it receives the token, then it immediately forwards the token to its neighbour. A process that requires the token waits until it receives it, but retains it. To exit the critical section, the process sends the token on to its neighbour.
- The arrangement of processes is shown in Figure. It is straightforward to verify that the conditions ME1 and ME2 are met by this algorithm, but that the token is not necessarily obtained in happened-before order.
- This algorithm continuously consumes network bandwidth: the processes send messages around the ring even when no process requires entry to the critical section.
- The delay experienced by a process requesting entry to the critical section is between 0 messages and N messages.
- To exit the critical section requires only one message. The synchronization delay between one process's exit from the critical section and the next process's entry is anywhere from 1 to N message transmissions.

3. With neat diagram explain Ring based Election Algorithm.

Ans:



Note: The election was started by process 17. The highest process identifier encountered so far is 24. Participant processes are shown in a darker tint.

- Each process p_i has a communication channel to the next process in the ring, $p_{(i+1) \bmod N}$, and all messages are sent clockwise around the ring.
- We assume that no failures occur, and that the system is asynchronous. The goal of this algorithm is to elect a single process called the coordinator, which is the process with the largest identifier.
- Initially, every process is marked as a *non-participant* in an election. Any process can begin an election. It proceeds by marking itself as a *participant*, placing its identifier in an election message and sending it to its clockwise neighbour.
- When a process receives an election message, it compares the identifier in the message with its own.
- If the arrived identifier is greater, then it forwards the message to its neighbour.
- If the arrived identifier is smaller and the receiver is not a *participant*, then it substitutes its own identifier in the message and forwards it; but it does not forward the message if it is already a *participant*.
- On forwarding an election message in any case, the process marks itself as a *participant*.
- If the received identifier is that of the receiver itself, then this process's identifier must be the greatest, and it becomes the coordinator.
- The coordinator marks itself as a *non-participant* once more and sends an elected message to its neighbour, announcing its election and enclosing its identity.
- When a process p_i receives an elected message, it marks itself as a nonparticipant, sets its variable $elected_i$ to the identifier in the message and, unless it is the new coordinator, forwards the message to its neighbour.

Unit – 5

Chapter – 9: Introduction to Cloud Computing

1. Define cloud computing and attributes for delivering computing services.

Ans:

Cloud computing is providing different IT services to customers over the Internet. It is the ability to deliver computing service over the Internet to the end-users on-demand or on a pay-as-you-go basis.

Attributes:

- Cloud computing uses Internet technologies to offer elastic services. The term elastic computing refers to the ability to dynamically acquire computing resources and support a variable workload. A cloud service provider maintains a massive infrastructure to support elastic services.
- The resources used for these services can be metered and the users can be charged only for the resources they use.
- Maintenance and security are ensured by service providers.
- Economy of scale allows service providers to operate more efficiently due to specialization and centralization.
- Cloud computing is cost-effective due to resource multiplexing; lower costs for the service provider are passed on to the cloud users.
- The application data is stored closer to the site where it is used in a device and location independent manner; potentially, this data storage strategy increases reliability and security and, at the same time, it lowers communication costs.

2. Discuss network centric computing and network centric content.

Ans:

a. Network-centric computing:
   - Focuses on the network as the central point of control and management
   - Allows for easy access and sharing of resources (such as storage and computing power) across multiple users and devices
   - Often used in cloud computing environments to improve scalability and performance
   - Allows for better resource utilization and cost-efficiency
   - Can improve security by centralizing access control and monitoring
   - Typical examples of large-scale network-centric systems are the **World-Wide Web and Computational Grids**.

b. Network-centric content:
   - Utilizes network-based resources (such as cloud storage and CDNs) to store, manage, and distribute content
   - Allows for easy access to content from any location and device
   - Improves performance and scalability by distributing content across multiple servers and locations
   - Can reduce costs associated with storage and distribution of large amounts of data.

- Allows for dynamic and adaptive content delivery based on network conditions and user preferences.
- Network-centric content is used in e-commerce to distribute and manage large amount of data such as images, videos, and product descriptions. Network-centric computing allows for efficient management of resources and infrastructure for online stores.

3. Explain different types of clouds with examples.

Ans:

1) Private Cloud

- Private cloud refers to a cloud deployment model operated exclusively to a single organization. It provides computing services to a private internal network and selected users, instead of the public in general.

Ex: HP Data Centers, Elastra-private cloud, Ubuntu, etc.

2) Public Cloud

- In this, the business rents the services that are required and pays for what is utilized on-demand. The resources are owned, maintained & operated by a third-party cloud service provider, and delivered over the internet.

Ex: AWS, GCP, Microsoft Azure, etc.

3) Community Cloud

- Community clouds are distributed systems created by integrating the services of different clouds to address the specific needs of an industry, a community, or a business sector. In the community cloud, the infrastructure is shared between organizations that have shared concerns or tasks. The cloud may be managed by an organization or a third party.

Ex: Our government organization within India may share computing infrastructure in the cloud to manage data.

4) Hybrid Cloud

- A hybrid cloud is a heterogeneous distributed system formed by combining facilities of public cloud and private cloud. For this reason, they are also called heterogeneous clouds.
- A major drawback of private deployments is the inability to scale on-demand and efficiently address peak loads. Here public clouds are needed. Hence, a hybrid cloud takes advantage of both public and private clouds.

Ex: Implementing database using on-premise private servers and using third party cloud service providers like AWS as load balancing or computing solutions.

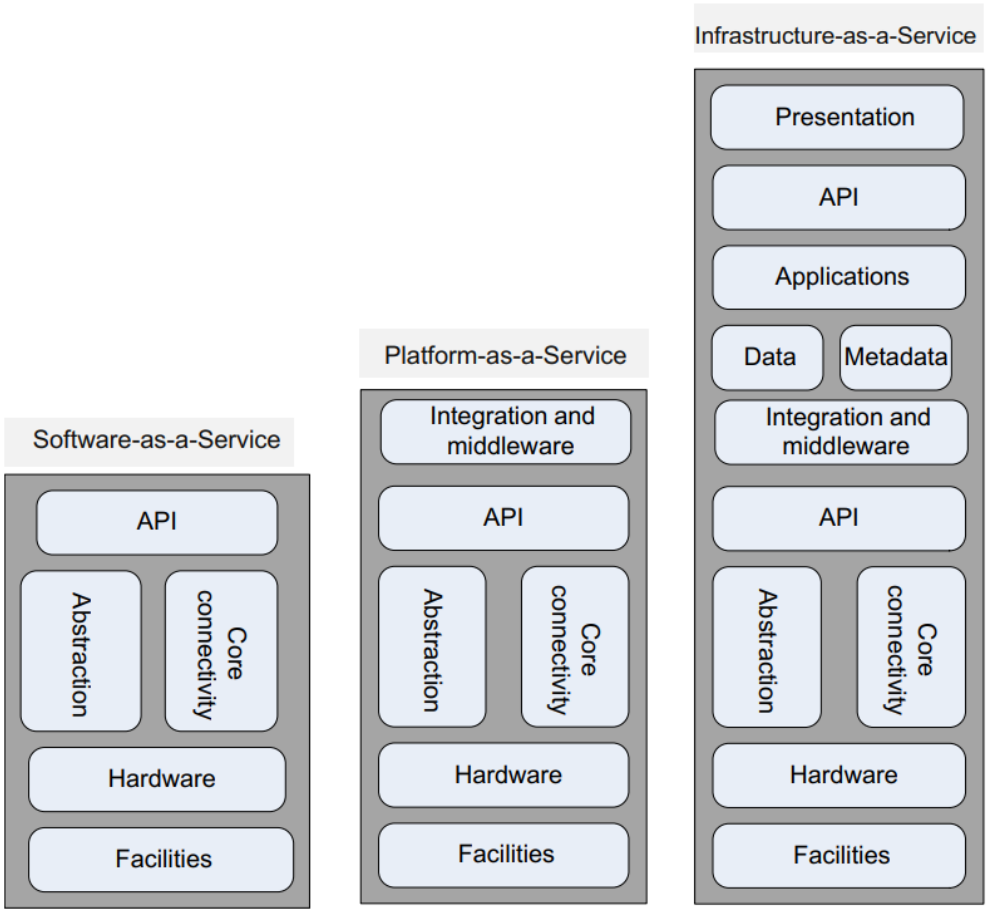
4. Discuss the success and failure of cloud computing.

Ans:

- **Advantages/Successes:**
  - **Flexibility:** Remote cloud servers offer almost unlimited bandwidth and storage space, which allows businesses to instantly scale up and down their capacities to support growth and cope when website traffic increases. This removes the need to purchase, configure and install equipment on-site.
  - **Business Continuity**: By investing in cloud computing, businesses can guarantee reliable disaster recovery and backup solutions without the hassle of setting them up on a physical device. Cost
  - **Efficiency**: The most significant advantage of cloud computing is the IT operational cost savings. Using remote servers removes the need for in-house storage equipment and application requirements, as well as overhead costs such as software updates, management, and data storage.
  - **Scalability and Performance**: Cloud technology is designed to be scaled to meet a business's changing IT requirements. As a company grows, it is inevitable that more storage space and bandwidth will be required to cope with increasing traffic to the website. Cloud servers can be deployed automatically to help businesses scale up and down and ensure optimum performance under heavy loads.
  - **Automatic Software Updates**: Many cloud service providers offer regular system updates to ensure IT requirements are consistently met. They ensure round-the-clock maintenance of cloud servers – including security updates.
- **Disadvantages/Failures:**
  - **Requires good speed internet with good bandwidth**: To access your cloud services, you need to have a good internet connection always with good bandwidth to upload or download files to/from the cloud
  - **Downtime**: Since the cloud requires high internet speed and good bandwidth, there is always a possibility of service outage, which can result in business downtime. Today, no business can afford revenue or business loss due to downtime or slow down from an interruption in critical business processes.
  - **Limited control of infrastructure**: Since you are not the owner of the infrastructure of the cloud, hence you don't have any control or have limited access to the cloud infra.
  - **Restricted or limited flexibility**: The cloud provides a huge list of services, but consuming them comes with a lot of restrictions and limited flexibility for your applications or developments. Also, platform dependency or 'vendor lock-in' can sometimes make it difficult for you to migrate from one provider to another.
  - **Ongoing costs**: Although you save your cost of spending on whole infrastructure and its management, on the cloud, you need to keep paying for services as long as you use them. But in traditional methods, you only need to invest once.

5. With the neat diagram explain cloud computing delivery models and services.

Ans:



i. Software as a Service (SaaS):

Software as a Service provides you with a completed product that is run and managed by the service provider. In most cases, people referring to Software as a Service are referring to end-user applications. With a SaaS offering you do not have to think about how the service is maintained or how the underlying infrastructure is managed; you only need to think about how you will use that particular piece of software. A common example of a SaaS application is web-based email where you can send and receive email without having to manage feature additions to the email product or maintaining the servers and operating systems that the email program is running on.

ii. Platform as a Service (PaaS):

Platforms as a service remove the need for organizations to manage the underlying infrastructure (usually hardware and operating systems) and allow you to focus on the deployment and management of your applications. This helps you be more efficient as you don't need to worry about resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running your application.

iii. Infrastructure as a Service (IaaS):

Infrastructure as a Service, sometimes abbreviated as IaaS, contains the basic building blocks for cloud IT and typically provide access to networking features, computers (virtual or on dedicated hardware), and data storage space. Infrastructure as a Service provides you with the highest level of flexibility and management control over your IT resources. Services offered by this delivery model include: server hosting, web servers, storage, computing hardware, operating systems, virtual instances, load balancing, Internet access, and bandwidth provisioning.

6. Discuss ethical issues encountered in cloud computing.

Ans:

- Cloud computing is based on a paradigm shift with profound implications for computing ethics. The main elements of this shift are:
  - The control is relinquished to third-party services.
  - The data is stored on multiple sites administered by several organizations.
  - Multiple services interoperate across the network.
- Unauthorized access, data corruption, infrastructure failure, and service unavailability are some of the risks related to relinquishing the control to third-party services; moreover, whenever a problem occurs, it is difficult to identify the source and the entity causing it.
- The complex structure of cloud services can make it difficult to determine who is responsible in case something undesirable happens. In a complex chain of events or systems, many entities contribute to an action, with undesirable consequences.
- Identity fraud and theft are made possible by the unauthorized access to personal data in circulation and by new forms of dissemination through social networks, which could also pose a danger to cloud computing.
- Cloud service providers have already collected petabytes of sensitive personal information stored in data centers around the world. The acceptance of cloud computing therefore will be determined by privacy issues addressed by these companies and the countries where the data centers are located.

-OR-

1. Privacy: Cloud computing involves storing personal data on remote servers, which can raise concerns about data privacy and security. Users may be concerned about the access and control that cloud providers have over their personal data.
2. Data security: Cloud computing also presents security risks, such as the potential for data breaches and unauthorized access to personal information.
3. Jurisdiction: Cloud providers may store data in different countries, which can create legal and ethical issues around data sovereignty and jurisdiction.
4. Transparency: Cloud providers may not be transparent about how they collect, use, and share user data, which can raise concerns about informed consent and user control over their personal information.
5. Interoperability: Some cloud providers may use proprietary standards and lock-in customers, making it difficult or impossible to move data and applications to other providers.

6. Dependence: Heavy dependency on cloud providers can lead to business continuity issues, if provider goes out of business or decide to stop providing certain service.
7. Environmental Impact: Cloud computing requires significant energy consumption and infrastructure, which can have a negative impact on the environment.
8. Digital divide: Limited access to cloud computing services can exacerbate existing social and economic inequalities, creating a "digital divide" between those who have access to these services and those who do not.

Overall, it's important for organizations and individuals to consider the ethical implications of cloud computing and take steps to mitigate any potential risks to personal privacy and security.

7. Explain cloud vulnerabilities.

Ans:

Clouds are affected by malicious attacks and failures of the infrastructure. Such events can affect Internet domain name servers and prevent access to a cloud or can directly affect the clouds.

- Misconfigured Cloud Storage: Cloud storage is a rich source of stolen data for cybercriminals. Despite the high stakes, organizations continue to make the mistake of misconfiguration of cloud storage which has cost many companies greatly. According to a report, nearly 70 million records were stolen or leaked in 2018 due to misconfigured cloud storage buckets.
- Insecure APIs: Application user interfaces (APIs) are intended to streamline cloud computing processes. However, if left insecure, APIs can open lines of communications for attackers to exploit cloud resources.
- Poor Access Management: Improper access management is perhaps the most common cloud computing security risk. When companies are not aware of how their employees are using cloud computing services, they could lose control of their data assets and ultimately become vulnerable to breaches and insider security threats.

Such events can affect the Internet domain name servers and prevent access to a cloud or can directly affect the clouds:

- In 2009, Google was the target of a denial-of-service attack which took down Google News and Gmail for several days.
- In 2012 lightning caused a prolonged down time at Amazon

8. Discuss the challenges faced by cloud computing.

Ans:

1. **Security**: One of the biggest challenges facing cloud computing is ensuring the security of data stored and processed on remote servers. This includes protecting against data breaches, unauthorized access, and other security threats.
2. **Data privacy**: Another challenge is ensuring that personal data is protected and kept private. This includes ensuring that cloud providers are transparent about how they collect, use, and share user data.
3. **Interoperability**: As different cloud providers use different technologies and standards, it can be difficult for organizations to move data and applications between different cloud environments.
4. **Compliance**: Organizations may have to comply with various laws and regulations regarding data storage, processing, and sharing. It can be challenging to ensure compliance with these regulations in a cloud computing environment.
5. **Reliability**: Cloud providers must ensure that their services are highly available and reliable, as downtime can have a significant impact on businesses and users.
6. **Governance**: Cloud providers must ensure that they have proper governance in place to manage the resources and services they offer.
7. **Cost:** Organizations must consider the cost of cloud computing, including the cost of data storage and processing, as well as the cost of migrating and managing applications and data in the cloud.
8. **Performance**: Organizations may experience performance issues, such as latency and bandwidth limitations, when using cloud computing services, particularly when accessing data from remote locations.
9. **Dependence**: Organizations may become too dependent on cloud providers, creating business continuity issues if the provider goes out of business or decide to stop providing certain service.